Simulink Code Generation

Tutorial for generating C code from Simulink Models using Simulink Coder

José Carlos Molina Fraticelli 4/25/2012

Contents

1.	Introduction	3			
2.	Good Programming Practices in Simulink	4			
3.	Choosing Variable Data Types	6			
4.	Hardware Implementation data type information	7			
5.	Naming Standards	7			
6.	Creation of practice model for code generation	8			
7.	Preparing practice model for code generation	. 17			
8.	Compiling and Executing generated code	. 29			
9.	Comparing output of MATLAB with that of the generated code	36			
Ref	eferences				
App	Appendix A Hardware Implementation Table				

1. Introduction

This document explains all the necessary steps in order to generate optimized C code from Simulink (Ref. [2]) Models. This document also covers some general information on good programming practices, selection of variable types, how to organize models and subsystems, and finally how to test the generated C code and compare it with data from MATLAB (Ref. [1]).

(NOTE: This guide was prepared using MATLAB/Simulink Ver. R2011b (Ref. [1]).)

2. Good Programming Practices in Simulink

Below are some guidelines to follow when creating Simulink (Ref. [2]) Models:

- Create bus objects for subsystems with many outputs and inputs. It makes organizing a model simpler by requiring less cabling and also makes management of the data contained in the bus easier when debugging as errors usually show up in a single location.
- Avoid using Mux blocks to create Bus signals. Always use a Bus Creator blocks.
- Avoid Crossing Cables when possible: There are a variety of ways to avoid crossing cables which can make debugging Simulink (Ref. [2]) models complicated. Some of these include: Organizing Subsystem Inputs to match cabling going into the subsystem, Goto and From blocks, Bus Creator blocks, Bus Selector blocks, among others.
- Determine variable types ahead of time. Simulink (Ref. [2]) defaults all data types to 64bit doubles by default. This can sometimes be a problem for code generation as not all targets can support these types of variables and it can be hard to change these once a model is created. This will save a lot of time trying to find problematic typecasts.
- Use MATLAB (Ref. [1]) structures when working with Bus Signals.
- Determine whether a model is a sub-function or a subsystem.
- Avoid Algebraic Loops. They are problematic for code generation as stated by Mathworks. If the algebraic loop cannot be avoided use a Unit Delay Block to break up the loop. (NOTE: Verify that adding this block has not changed the output of the model.)
- Avoid using different sampling times between models. This is not supported by code generation.
- Avoid using variable-step solvers. The only valid solver for code generation is the Fixed Step Solver with a discrete time step.
- Build models from the ground up with code generation in mind.
- Test models as a standalone piece both in simulation and in the generated code and ensure it is working before adding additional models.
- Avoid using equal names for buses, data, constants and models. This can cause serious problems with code generation. Use a naming standard to clearly define what every object is or isn't.
- Rename models such as "Foo One Model"; to Foo_One_Model. This will help avoid problems with code generation.

- If a model name is too long try shortening it. Model names should not exceed 20 characters.
- Align models and objects that are part of the model using the alignment tools available in Simulink (Ref. [2]) these can be accessed by selecting two or more model objects and selecting align blocks. This can help clean up models immensely
- Make sure the warning indicator on MATLAB (Ref. [1]) function blocks is green. Try to fix all warnings pointed to by this indicator. Also always add the %#codegen pragma directive below the function name declaration and before any code.
- Don't use Goto and From blocks between different models. Use a signal line instead.
- Avoid the use of extremely memory intensive blocks such as 'Fuzzy Logic controllers' if generating code for embedded platforms such as microcontrollers; as the generated code will not fit in the stack due to the high amount of floating point variables required.
- Avoid excessive unnecessary variables. Memory is at a premium in embedded hardware as opposed to a desktop computer.
- Use consistent signal names. This can help ease debugging immensely.
- Align all inputs to the left of the model and all outputs to the right of the model. This can help find problematic inputs/outputs faster.

3. Choosing Variable Data Types

In order to determine the correct data type for a variable one must determine what that variable will be used for. Simulink (Ref. [2]), by default, will generate all variables as 64-bit doubles. This can cause problems for embedded platforms with a low memory footprint such as microcontrollers which is where generated code is usually targeted to. Below are some tips in order to determine which variable type is viable for each action:

- boolean: If the variable is to be used as a ON/OFF switch then this variable type should be used
- int8: If the variable does not exceed 8 bits and is a character then this variable type should be used
- int16: If the variable does not exceed 16 bits and is an integer then this variable data type should be used
- int32: If the variable does not exceed 32 bits and is an integer then this variable data type should be used
- single: If the variable is a single precision floating point number then this variable data type should be used
- double: If the variable is a double precision floating point number then this variable data type should be used. (NOTE: This is the default variable type for Simulink (Ref. [2]))
- If the variables to be defined contain many elements then a structure must be created to house these elements. A Bus must be created from said structure and then used as the data type of the variable.

4. Hardware Implementation data type information

Below is a table listing device specific information for Hardware Implementation of the Simulation and Generated code. This information is set in the Hardware Implementation pane located in the Simulation Configuration Parameters:

Device	Number of bits							Lar	Largest atomic size		Byte	Rounds	Shift
vendor / Device type	cha	r	short	in	t loi	ng r	nativ	e int	float	ordering		to	right
ARM Compatible													
ARM 7/8/9/10	8	16	32	32	2	32		Long	Float	X		X	x
Microchip													
dsPIC	8	16	16	32	2	16		X	x	Li	ttle Endian	Zero	Set
Texas Instruments													
MSP430	8	1	6 16	5	32		16	j	X	X	Little Endian	Zero	Set

Table 1: Hardware Implementation Partial

(NOTE: For a complete Hardware Implementation table refer to Appendix A.)

5. Naming Standards

To remain consistent on which object is what, and also to make debugging of problems easier, it is important to establish a naming standard. This naming standard must be applied to models, constants, variables and buses. An example of this standard is listed below and can be used as is or with some modifications if required:

- If the object in question is a model then the name of that model should be in the form of: "Name_Model"
- If the object in question is a constant then the name of that constant should be in the form of: "Name_Constant"
- If the object in question is a bus then the name of that bus should be in the form of: "Name_Bus"

- If the object in question is a signal then the name of that signal should be in the form of: "Name_Signal"
- If the object in question is a function then the name of that signal should be in the form of: "Name_Func"
- If the object in question is a subsystem input then the name of that input should be in the form of: "Name_Input"
- If the object in question is a subsystem input then the name of that input should be in the form of: "Name_Output"

6. Creation of practice model for code generation

In this section a new practice model will be created in order to demonstrate the necessary steps to do successful code generation from any Simulink (Ref. [2]) model.

Open up MATLAB (Ref. [1]) and type the 'buseditor' command on the command window in order to access the Bus Editor.



Bus Editor GUI:



Once here click on File > Add Bus, (or alternatively Ctrl+B), while having the Base Workspace selected. A new Bus Object will be created in the MATLAB (Ref. [1]) workspace. Rename this Bus Object to Inputs by modifying the Name Field in the Bus Editor:

Simulink.	Bus: Inputs						
Propertie	s						
	-						
Name:	Inputs						
Code g	eneration options						
DataSo	ope: Auto 🔻						
Header	File:						
Alignme	ent: -1						
Descripti	on:						

Click Apply at the bottom of this window.

Now add two new elements to this Bus by going to File>Add/Insert Element (or alternatively Ctrl+E) while having this Bus Object selected. These elements will be called Input_A and Input_B respectively and will be of type int32:

Simulink.BusElement: Input_A
Properties
Name: Input_A
DataType: jint32 <<
Data Type Assistant
Mode: Built in 💌 int32 💌 Data type override: Inherit
Minimum:
Maximum:
Complexity: real
Dimensions: 1
DimensionsMode: Fixed
SamplingMode: Sample based
SampleTime: -1

Now repeat the previous process for the Outputs Bus but this time add five elements sum of type int32, sub of type int32, mul of type int32, div of type double, and sum_gain of type int32. Once finished the Bus Editor Window should contain the following objects:



Go back to the workspace and write the following command in the command window: 'gain = int32(1)'. You should see the following now:



Now that the workspace contains all necessary bus objects and variables with their defined data types proceed to create a new model for Simulink (Ref. [2]) by going to File>New>Model at the top toolbar:

1	MATL	AB R201	1b					
Fil	e Edit	Debug	Parallel	Desktop	Wir	ndow	Help	
	New				∢	Scr	ript	Ctrl+N
	Open.			Ctrl+0	С	Fu	nction	
	Close	Command	Window	Ctrl+\	N	Cla	iss	
	Import	Data				Eni	umeration	
	Save V	Vorkspace	As	Ctrl+9	5	Fig	jure	
—					_	Var	riable	
	Set Pa	th				Mo	del	
	Preferences					GU		
	Page S	ietup				De	ployment Project	
	Print			Ctrl+F	•	Co	de Generation Projec	:t
	Print S	election				Sim	nulink Project	
	1 D:\	.TSAT_ER	T\driveSTI	<.m				
	2 C:\	.ibrary_FS	; W\busDel	.m				
	з с:∖	.ro_mod\I	nDataFile.	m				
	4 ⊂:\	mod\Cu	besat_init	.m				
	Exit M	ATLAB		Ctrl+0	Ş			
	🛃 Su	um_sfun.m	iexw32			-		

Now this new model should be open:



Bring up the Simulink (Ref. [2]) Library Browser and add the following blocks to the model:

• 3 Bus Creator Blocks

- 3 Constant Blocks
- 7 Bus Selector Blocks
- 2 Display Blocks
- 5 MATLAB (Ref. [1]) Function Blocks
- 1 Model Info Block

Once these are added to the model organize them in the following manner:



Make sure that the Output data types of the Input_A, Input_B and gain constants are of type int32. (NOTE: Set the gain constant value to 'gain')

Open the first of the MATLAB (Ref. [1]) Function blocks and write the following code inside:

sum_func:



For the next four blocks write the following code in order from top to bottom:

sub_func:



mul_func:



div_func:



sum_gain_func:

HATLAB Function Block Editor - Block: CodegenT_Model/sum_gain_Func	
<u>File Edit Text Debug Tools Window H</u> elp	× 5
🗋 😅 🛛 🍐 ங 🛍 🤊 😋 🖊 📰 🛃 💿 🛧 🛛 🥸 🕮 👘	* 🗖 💌
1 function sum_gain= sum_gain_func(Input_A, Input	B,gai
2 %#codegen	
3 - sum_gain = int32(Input_A+Input_B+gain);	
4	
Ready Ln 1 Col 1	1

Connect the model as shown below:



(NOTE: Make sure the First Bus Creator on the left has a data type of 'Bus: Inputs' and that the following Bus Creator to the right has a datatype of 'Bus: Outputs' as can be seen on the above. Also be sure to label all signal cables with their corresponding names)

Configure the Model Info Block with the following tokens then click Apply and OK:

📣 Model Info: untitled	
Model properties: Modified Date Modified Comment Model Version Model Name Description Last Modified By Last Modification Date Configuration manager properties: >	Enter text and tokens to display on Model Info block: Model Info % <created> %<creator> %<modifieddate> %<modifiedcomment> %<modelversion> %<modelversion> %<description> %<lastmodifiedby> %<lastmodificationdate></lastmodificationdate></lastmodifiedby></description></modelversion></modelversion></modifiedcomment></modifieddate></creator></created>
Horizontal text alignment: Center	Show block frame
OK Cano	cel Help Apply

Run the model and look at the output seen on the display block; if the Input Constant values of Input_A and Input_B are set to 1 and 2 respectively then the output of the simulation should be um = 3, sub = -1, mul = 2, $sum_{gain} = 4$ and div = 0.5:



Select this section of the model and right click it. Afterwards select Create Subsystem from the popup menu:



The model should now look as follows. Save this model as CodegenT_Model:



7. Preparing practice model for code generation

Now that the CodegenT_Model is finished and running it is time to prepare it for code generation. To start right click on the Subsystem Model and select Subsystem Parameters:

	🙀 Function Block Parameters: Subsystem						
	Subsystem						
	Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select 'Treat as atomic unit'.						
	Main Code Generation						
	Show port labels FromPortIcon						
	Read/Write permissions: ReadWrite						
	Name of error callback function:						
	Permit hierarchical resolution: All						
→ In2 Subsystem	Treat as atomic unit						
	OK Cancel Help Apply						
	<div> Display1</div>						

Check the box that is labeled "Treat as atomic unit"; click Apply then OK. This will change the borders of the Subsystem to a bold line indicating that the system is now atomic and will execute as a single unit inside the simulation. Systems must be atomic in order to be generated as re-usable C functions using Real-Time Workshop. The Subsystem model should now look as follows (NOTE: Changing a Subsystem to an atomic representation can change simulation output; always verify that the simulation is still working before generating code from it.):



Rename the Subsystem model to Calc_Test_Model:



Once again right click on the model and open the Subsystem Parameters, go to the Code Generation Tab which is enabled when the Subsystem is declared as atomic and choose the "Reusable Function" option on the "Function packaging" dropdown list. Afterwards two more dropdown lists will appear "Function name options" and "File name options"; For the first dropdown list choose "User Specified" and write the following name in the provided text space "calc_functions". Afterwards choose the "Use function name" option for the "File name options" dropdown list. Finally click Apply then OK to close the subsystem parameters:

Function Block Parameters: Calc_Test_Model									
Subsystem									
Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select 'Treat as atomic unit'.									
Main Code Generation									
Function packaging: Reusable function									
Function name options: User specified									
Function name:									
calc_functions									
File name options: Use function name									
OK Cancel Help Apply									

Next go to the top toolbar in Simulink (Ref. [2]) and click on Simulation > Configuration Parameters. The following window will open:

Configuration Parameters: Cod	legenT_Model/Config	guration (Active)			×	
Select:	-Simulation time-				<u>^</u>	
····Solver ····Data Import/Export	Start time: 0.0		Stop time: 10.0			
• Optimization • Diagnostics	Solver options					
	Туре:	Variable-step 🔹	Solver:	ode45 (Dormand-Prince) 🔹		
	Max step size:	auto	Relative tolerance:	1e-3	=	
	Min step size:	auto	Absolute tolerance:	auto	_	
	Initial step size:	auto	Shape preservation:	Disable all 🔹		
	Number of conse	cutive min steps:	1			
	Tasking and sam	ple time options				
	Tasking mode for	periodic sample times:	Auto			
	Automatically	handle rate transition for data trans	sfer			
	Higher priority	value indicates higher task priority				
	Zero-crossing op	tions				
	Zero-crossing cor	ntrol: Use local settings	 Algorithm: 	Nonadaptive		
	Time tolerance:	10*128*eps	Signal threshold	: auto		
•				1000	*	
0			<u>0</u> K	Cancel Help	pply	

These are all the available properties for the simulation. For code generation only some of these properties need to be modified. To start off set the Solver pane options as shown below:

Solver							
Data Import/Export	Start time: 0.0	Stop time: inf					
Optimization Diagnostics	Solver options	Calver diamete (na continuous states)					
Model Referencing	Type: Hixed-step	Solver: discrete (no continuous states)					
Simulation Target Code Generation	Fixed-step size (fundamental sample time):	1					
	Tasking and sample time options						
	Periodic sample time constraint:	Unconstrained					
	Tasking mode for periodic sample times:	Auto					
	Automatically handle rate transition for data transfer						
	🔲 Higher priority value indicates higher task priority						

Next in Data Import/Export pane change the signal logging name to BusData (NOTE: Signal logging can be used for debugging and generating files from simulation output data):

🍓 Configuration Parameters: Cod	legenT_Model/Config	juration (Active)			—X —	
Select:	- Load from works	pace				
Solver Data Import/Export	Input:	[t, u]				
Optimization Diagnostics]				
Hardware Implementation	Save to workspa	ce				
Model Referencing	Time, State, Ou	itput				
	☑ Time:	tout	Format:	Array 🔻	E	
	States:	xout	Limit data points to last:	1000		
	Output:	yout	Decimation:	1		
	Final states:	xFinal	Save complete SimState	in final state		
	Signals					
	🗵 Signal loggir	ng: <mark>BusData</mark> S	gnal logging format: ModelDa	taLogs 🔻		
	Configure Sig	nals to Log				
	Data Store Men	nory				
	☑ Data stores:	dsmout				

For the Optimization pane change the Application lifespan setting to inf:

🍓 Configuration Parameters: Cod	odegenT_Model/Configuration (Active)	×
Select: 	Simulation and code generation Simulation and code generation Conditional input branch execution Conditional input branch execution Implement logic signals as Boolean data (vs. double) Application lifespan (days) inf Use integer division to handle net slopes that are reciprocals of integers Use floating-point multiplication to handle net slope corrections Code generation Data initialization Use memset to initialize floats and doubles to 0.0 Integer and fixed-point Remove code from floating-point to integer conversions that wraps out-of-range values Remove code from floating-point to integer conversions with saturation that maps NaN to zero Accelerating simulations Compiler optimization level: Optimizations off (faster builds) Verbose accelerator builds	E

Jump over to the Hardware Implementation pane and set the settings of this pane as outlined in Table 1 based on the Hardware for which the code will be generated (NOTE: For this example Unspecified (32-bit generic) will be used):

Select:	Embedded hardware (simulation and code generation)	
Solver	Device vendor: Generic Device type: Unspecified (assume 32-bit Generic)	
••• Optimization	Number of bits Largest atomic size	
Diagnostics Hardware Implementatio	char: 8 short: 16 int: 32 integer: Char	
Model Referencing Simulation Target Code Generation	long: 32 float: 32 double: 64 native: 32 pointer: 32 floating-point: None	
	Byte ordering: Unspecified Signed integer division rounds to: Undefined Signed integer division rounds to: Undefined	
	Emulation hardware (code generation only)	
	V None	

Go to the Code Generation Pane and select the following options:

🆏 Configuration Parameters: Cod	degenT_Model/Configuration (Active)	×
Select:	Target selection	
Solver Data Import/Export Optimization Optimization Add Referencing Simulation Target Code Generation "Report Comments "Symbols "Custom Code "Debug "Interface "SIL and PIL Verification "Code Style	System target file: ert.tlc Browse Language: C ▼ Description: Embedded Coder ▼ Build process Compiler optimization level: Optimizations off (faster builds) ▼ ▼ TLC options: Image: Image: Image: Image: Makefile configuration Image: Image: Image: Image: Make command: Imake_rtw Image: Image:	Ħ
Templates Code Placement Data Type Replacem Memory Sections	Data specification override Ignore custom storage classes Ignore test point signals Code Generation Advisor Prioritized objectives: Execution efficiency, ROM efficiency, RAM efficiency, Set objectives: On (stop for warnings) Check model before generating code: On (stop for warnings) ✓ Generate code only	

Set Objectives Pop-Up to the desired objectives:

Set Objectives - Code Generation Advisor Description Select and prioritize your code generat objectives, for details, see the docume	r tion objective entation.	s. You can add custom	X
Available objectives Debugging MISRA-C:2004 guidelines	+ +	Selected objectives - prioritized Execution efficiency ROM efficiency RAM efficiency Traceability Safety precaution	1
,		OK Cancel	Help

Make sure to select the correct target file based on the desired target. For this example ert.tlc Embedded Coder will be used.

Select the Report Pane and set the following options:

🍓 Configuration Parameters: Cod	legenT_Model/Configuration (Active)		—
Select: 	 Create code generation report Navigation Code-to-model Model-to-code Configure Traceability Report Contents Eliminated / virtual blocks Traceable Simulink blocks Traceable Stateflow objects Traceable MATLAB functions 	✓ Launch report automatically	E
Interface SIL and PIL Verification Code Style Templates Code Placement Data Type Replacem Memory Sections	Metrics		
0		<u>Q</u> K <u>C</u> ancel <u>H</u>	elp <u>A</u> pply

Go to the Interface pane and select the following options:

	Coffuerra anuironna at			
select:	Software environment			
™Solver ™Data Import/Export ™Optimization	Target function library: Shared code placement:	C89/C90 (Al Auto	NSI)	▼ Custom
Diagnostics Hardware Implementation Model Referencing Simulation Target	Support: I floating-point I absolute time	numbers	non-finite numbers continuous time	Complex numbers
Code Generation Report Comments	Multiword type definitions:	signals System de	fined	•
- Symbols	Code interface			
- Debug - Interface	GRT compatible call inte	erface 🗵 S	ingle output/update function	Terminate function required
SIL and PIL Verification Code Style Templates	✓ Generate reusable code	Reus Pass	sable code error diagnostic: s root-level I/O as: Individua	Error I arguments
Code Placement	Generate preprocessor co	nditionals: 🛛	Jse local settings	•
Data Type Replacem Memory Sections	Configure Model Function	n real-time n	nodel data structure	Combine signal/state structures
	Data exchange			

Make sure the "Generate reusable code" checkbox is checked in this pane.

Go to the Code Style pane and check all checkboxes. This is done to set the coding style as humanly readable as possible for code generation:

🍓 Configuration Parameters: Co	degenT_Model/Configuration (Act	ive)		×
Select:	Code templates			<u> </u>
Solver Data Import/Export	Source file (*.c) template:	ert_code_template.cgt	Browse Edit	
Optimization Diagnostics	Header file (*.h) template:	ert_code_template.cgt	Browse Edit	
	Data templates			
	Source file (*.c) template:	ert_code_template.cgt	Browse Edit	E
⊡ Code Generation …Report	Header file (*.h) template:	ert_code_template.cgt	Browse Edit	
Comments Symbols	Custom templates			
Custom Code	File customization template:	example_file_process.tlc	Browse Edit	
····Debug ····Interface	🔽 Generate an example ma	in program		
SIL and PIL Verification	Target operating system:	BareBoardExample	•	
Code Style				
Code Placement				
Data Type Replacem				
Memory Sections				
				-
0			OK <u>C</u> ancel <u>H</u> elp	<u>A</u> pply

Next go to the Templates pane and set the following options:

In this pane source code generation templates may be selected. This requires advanced knowledge of tlc language. Information on tlc language and files can be found in the MATLAB (Ref. [1]) help window.

Once again return to the Code Generation pane; click Apply then click the generate code button. A model check procedure will now start to verify that all settings are correct for code generation:

🕼 Code Generation Advisor - CodegenT_Model
File Edit Run View Help
S Find: name and description
 Code Generation Objectives Code Generation Objectives Check model configuration settings ag Check for optimal bus virtuality Check Data Store Memory blocks for m Aldentify block output signals with cor Identify questionable blocks within the Check thardware implementation Identify questionable software environ Identify questionable software environ Identify questionable software environ Identify questionable software environ Analysis Code Generation Objectives Show report after run Report Report Report Report Report Report Report 402.html Save As Date/Time: 10-Apr-2012 22:09:37 Summary: Pass: 8 Fail: 0 k Warning: 6 Not Run: 0 Tips To process all enabled items in this folder. To process all enabled items in this folder. To automatically display the report after processing, select "Show report after run". To display the last report generated, dick the "Report" path link.
Model Explorer Initialized

Select the first warning symbol on the top of the Code Generation Objectives list and then click on Modify Parameters:

	Check model configuration settings against code general	tion objectives		
Code Generation Objectives	check model comparation sectings against code genera	and objectives		
🛽 🥼 Check model configuration settings				
🛛 🥝 Identify unconnected lines, input p	The following parameter values are not optimized for the select	ed objectives: Execu	tion efficiency, ROM	
Check for optimal bus virtuality	endency, whereindency, naceability, sarety precadion.			Ξ
🛕 Check Data Store Memory blocks for m	To automatically fix the warning, click the 'Modify Parameters' b	utton and then rerun	the check. To manually fix	
^Identify block output signals with cor	the warning, click the parameter hyperlink to open the Configur apply the recommended value	ation Parameters dia	log box, and manually	
Identify questionable blocks within the	(Objectives: Execution efficiency, ROM efficiency, RAM efficien	cy, Traceability, Safet	ty precaution)	
Check the hardware implementation	Parameter	Current Value	Recommended Value	
Identify questionable software environ	Suppress error status in real-time model data structure	off	on	
Identify questionable code instrumenta	non-finite numbers	on	off	
Check for blocks that have constraints	Compiler optimization level	Off	on	
^Identify guestionable subsystem setting	Minimize data copies between local and global variables	off	on	
A Identify blocks that generate expensiv	Inline parameters	off	on	
^Identify guestionable fixed-point ope	Remove code from floating-point to integer conversions that	off	on	Ŧ
Check for efficiency optimization param				
	Action			
	Change current values to recommended value. Some parameter	s might require manu	al changes.	
	Modify Parameters			

This will set the conflicting parameters to the correct settings for code generation. Once this is done click on Run This Check will in the same window:

	Check model configuration settings against code generation object	tives		
🛛 🙀 Code Generation Objectives				
📝 🥝 Check model configuration settings	Anarysis			Ē
📝 🤣 Identify unconnected lines, input p	Check model configuration settings against the code generation objectives.	Successfully passing this check may t	take	
📝 🥝 Check for optimal bus virtuality	multiple iterations since a change to one option can impact other options.			
📝 🛕 Check Data Store Memory blocks fo	Run This Check			
📝 🥝 ^Identify block output signals with				Ξ
📝 🤣 Identify questionable blocks within	Result: 📀 Passed			
📝 🛕 Check the hardware implementation				
📝 🛕 Identify questionable software envi			A	
📝 🥝 Identify questionable code instrum	Passed (Objectives: Execution efficiency, ROM efficiency, RAM efficiency, Traceabi	lity. Safety precaution)	=	
📝 🤣 Check for blocks that have constraii	The following parameters have been checked and confirmed with the recom	imended value		
📝 🛕 ^Identify questionable subsystem s	Parameter	Value		
📝 🛕 ^Identify blocks that generate expe	non-inlined S-functions	off		
📝 🥝 Aldentify questionable fixed-point	Suppress error status in real-time model data structure	on		
📝 🥝 Check for efficiency optimization pa	MAT-file logging	off		
	GRT compatible call interface	off		
	continuous time	off		
	non-finite numbers	off		
	Single output/update function	on		
	Minimize algebraic loop occurrences	off		Ŧ

Do the same procedure for the remaining warnings if necessary based on individual model configurations. In the case of this simple example it is not necessary to fix all warnings as they are only there to prevent what could be future issues but not really simulation errors.

Once done here go back to the Configuration Parameters and change the "Check model before generating code" dropdown list to the "Off" setting in order to bypass the model advisor. Click Apply then OK to save these changes then close the Configuration Parameters:

Code Generation Advisor	
Prioritized objectives: Execution efficiency, ROM efficiency, RAM efficiency,	Set objectives
Check model before generating code: Off	Check model
Generate code only	Generate code

Right click on the Calc_Test_Model Subsystem and go to Code Generation > Build Subsystem. If there are not conflicts the following window should appear. (NOTE: It is possible an error message might pop-up due to an unset or incorrectly set configuration parameter option. This is normal and can be easily fixed by changing the conflicting parameter):

A Build code for Subsystem:Calc_1	Test_Model			×
Pick tunable parameters				
Variable Name		Class	StorageClass	
🛨 gain		int32	Inlined	<u> </u>
				=
Blocks using selected variable -				
Block		Parent		
				~
Ctatue		Build	Cancel Hel	p
Select tunable parameters and o	click Build			

This window contains all workspace constants that are to be used in the generated code and offer storage options for the generated code. In the case of this example the variable gain can be left as "Inlined". Click on Build and the code generation process will start.

The code generation progress should be shown in the command window as seen below:



If all goes well the following Code Generation Report should show up:



From here the generated code can be evaluated and can also be traced back to the Simulink (Ref. [2]) Model. For more information on the Code Generation Report consult the MATLAB (Ref. [1]) help window.

Now on the current MATLAB (Ref. [1]) path there should be two folders called Calc_Test_Model_ert_rtw and slprj respectively:

Name	Date modified	Туре
퉬 Calc_Test_Model_ert_rtw	4/13/2012 5:21 PM	File folder
퉬 slprj	4/13/2012 5:06 PM	File folder
a while a second sec	4/13/2012 5:10 PM	Microsoft (
🔺 busdefines.m	4/5/2012 3:35 PM	M File
Calc_Test_Model_sfun.mexw64	4/13/2012 5:20 PM	MEXW64 F
CodegenT_Model.mdl	4/10/2012 10:23 PM	MDL File

These folders contain all the generated code from the Simulink (Ref. [2]) Model; copy and paste these folders to any desired location for the next section of the tutorial where the code will be compiled and executed.

8. Compiling and Executing generated code

Now that the code is generated the next step is to test the code by compiling it and creating an executable to run it. For this section of the tutorial a C IDE and compiler are required. This tutorial will use Bloodshed Software's Dev-C++ IDE [2] but these steps can be done in any IDE with any compiler. Dev-C++ (Ref. [3]) can be downloaded from Bloodshed Software's webpage: http://www.bloodshed.net/download.html

VEW Pavorees	Taolo Help		
😸 🔛 IEE-75	4 Analysis 🕼 Floating Fact to Hex Conver. 🔺 Hardware Implementation P 🎓 Namitral Subsystem Code 🐥 Das Copies Explaned: Gay 🛊 Senderk.Bas. readeObject 🐥 Defining Data Stores with Su. 💽 Sognated Store * 🔊 Web St	ice Galery * 💋 Hone (MSPC Software Dev	
oftware - Downlo	**	 N • ⊡ - ≕ ⊕ •	Page = Safety = Tool
000	Providing Free Software to the Internet comm	unity	
	Dev C++		Get
			Bloodsh
WH CD	Biological Car-Core a standards respirated bevelopment Environment (CR) for the CC++ programming anguage, it uses integra part of CCC (CRIC Conpair Calendor) as the conpair. Calendor) as the conpair Calendor) as the conpa	sed company.	CD (
	Free Saftware (Urit) Far Visoure 82, 83, 647, 2000, XP		
	Authors : Colin Laplace, Mike Berg, Hospi Lai : Development		Get it on Blog
ers ers liet	Minger campiler: Kumit Khan, Jan Jaap van der Heiden, Calin Hendrik and GAU coders.		CD
			You can get all the software available
	Dev Pakela E Al Antonio Internated Analonanet Analonanet (PC), which is able to create Workson or consults Asset Dental International Texts Based International Paral Parad Paral Parad Parad Paral Parad Parad Parad Parad Parad Parad Parad Parad Parad Pa		site on a CD. This of all software in
	Let-result is a fail-resulted integrated entropyment environment (Let, minor is under a under environment and grade state provide result of operations and under environment and a state of under		ONE CD:
	The particular III Free Vision Vision State Stat		Dev-C++ 5 beta
braission	Autors :		Dev-C++ 4.01 up
ce	Coin Layse: Man Development Hong Lay IC Development Annual Control Co		Dev-Pascal 1.9
			Avi Creator 1.0
	QuickInstall 2		Dev-C++ 4 source Dev-Pascal 1.9 s
	This program creates powerful installations for your software. Very easy to handle and powerful.		Muttox 4.0 Fast Cleaner 1.0
	Freel		LaserWar source
	Fel Vinolovia los sindi se Autoro: Colo Lugaice		Delphi Calo-By-Step 1.0
			Dev-C++ Packap + Goodes 1
	An Ocean Alfo Table Internet Control and Barless sciences and Barles alf You can also some moles the antibioties alfunct Alfon		_
	From Terr Vindows 55, 56, MT and 2000		
	Adro : Uni Lajace		
	Boodshed Antiquities		
	Sone att software writes years app :) meyte you'll fod some usefulfumy.)		
	Some all actuare writer years age) regte you'l find some vaetafforey.)		Copyright Blog
	Some of surface writer years app.) regiser you? The some same fullying .		Crowinght Blog Software

Download the Dev-C++ (Ref. [3]) setup and install it by following the on-screen prompts:



Once finished with the installation open up the Dev-C++ (Ref. [3]) IDE:

DevrC++4392	_ 6 ×
5 dit Sarch free Projet Deoute Debug Tools COS Window Heip	
vev [usite] [usite] [usite]	
R Congate May Resource Mill Conget Leg & Debug G. Free Result	
[Peadu	

Go to the top toolbar then go to File > New > Project, afterwards select the following options: Empty Project, check C Project and Name the project Calc_Test_Generated_Model:

New project					— ×-
Basic Introd	luction MultiM	edia			
		4.00	<u> </u>	U.	
Windows Application	Console Application	Static Library	DLL	Empty Project	
Description: An empty pro	oject				
 Project optior 	ns:				
Name: Calc_Test_Ge	enerated_Model			∑ Project ∑ <u>M</u> ake Default L	○ C <u>+</u> + Project anguage
			✓ <u>0</u> k	X Can	cel ? <u>H</u> elp

Click on Ok and a Prompt will come up asking for a location to save the project. Choose any desired location. (NOTE: It is recommended to create a new folder for this project)

After creating the project the left hand tab will contain a root folder with the name of the project. This is where all source code will be located:

W Dev-C++ 4.9.9.2
File Edit Search View Project E
💹 🍕 🗖 📰 🖬 📲 🛲 🏎
🗄 🗖 🖪 🎛 🖌 🤋 🥹 🕻
Project Classes Debug

Now drag the copies of the Calc_Test_Model_ert_rtw and the slprj folders into the Calc_Test_Generated_Model project folder in the previously selected location:

	Calc Test	Gener	ated Model 🕨					
Organize 🗸	🕞 Open	In	clude in library 🔻	Share with 🔻	Burn	New folder	≡ ▼ 🚺	0
🛛 😭 Favorites			Name	^		Date modified	Туре	Size
📃 Desktor)		퉬 Calc_Test_Mo	del_ert_rtw		4/13/2012 5:21 PM	File folder	
📃 🚺 Downlo	ads		퉬 slprj			4/13/2012 5:06 PM	File folder	

Next go to the top toolbar then go to Project > Project Options then in the Pop Up window go to the Directories Tab then inside of it go to the Include Directories Sub Tab. Add the following paths to the include directories: ".\Calc_Test_Model_ert_rtw" and ".\slprj\ert_sharedutils" (NOTE: replace the dot with the absolute path to the Calc_Test_Generated_Model folder):

Project Options	x
General Files Compiler Parameters Directories Build Opti Makefile Version i	nfo
Library Directories Include Directories Resource Directories	
C:\Users\Jose\Desktop\Calc_Test_Generated_Model\Calc_Test_Model_ert_rtw C:\Users\Jose\Desktop\Calc_Test_Generated_Model\s prj\ert_sharedutils	
	2
Replace Add Delete Invalid	
🚺 🖌 🖸 🖉 🔤	p

Right click on the Calc_Test_Generated_Model folder on the left hand tab and then click on "Add Folder" call the folder Calc_Test then click OK:

Add folder	×
Enter the new folder's name:	
Calc_Test	
OK Cancel	

Add an additional folder to the top folder called utilities:

🔐 Dev-C++ 4.9.9.2 - [Calc_Test_Gene
File Edit Search View Project E
🥘 🍓 🗖 📰 🖬 📲 📕 🏎
🔠 🗖 🖪 🎛 🖌 🤋 🥹 [
Project Classes Debug
□- Unit Calc_Test Generated_Model Calc_Test Calc_Test Utilities

Right click on Calc_Test and then click on "Add to project". Browse to the Calc_Test_Model_ert_rtw folder then add all .c and .h files excluding the ert_main.c. Repeat this for the utilities folder by going to slprj\ert_sharedutils. Finally right click on the top Calc_Test_Generated_Model folder and add the ert_main.c from the Calc_Test_Model_ert_rtw folder:

Look in:	Calc_Test_Model_ert_rtw	← 🗈 📸 ᠇	
œ.	Name	Date modified	Туре
et al and a second seco	🌗 html	4/13/2012 5:21 PM	File fo
Recent Places	calc_functions.h	4/13/2012 5:21 PM	C Hea
	Calc_Test_Model.h	4/13/2012 5:21 PM	C Hea
Desktop	Calc_Test_Model_private.h	4/13/2012 5:21 PM	C Hea
	Calc_Test_Model_types.h	4/13/2012 5:21 PM	C Hea
1000 A	alc_functions.c	4/13/2012 5:21 PM	C Sou
Libraries	Calc_Test_Model.c	4/13/2012 5:21 PM	C Sou
	ert_main.c	4/13/2012 5:21 PM	C Sou
	🔊 buildInfo	4/13/2012 5:21 PM	Micro
Computer	🔊 codeInfo	4/13/2012 5:21 PM	Micro
	Calc_Test_Model.mk	4/13/2012 5:21 PM	MK Fi
Network	Calc_Test_Model_ref.rsp	4/13/2012 5:21 PM	RSP Fi
Hotmont	defines.txt	4/13/2012 5:21 PM	Text D ▶
	File name: "calc_functions.h" "Calc_Test_Mod	del.h" "Calc. 💌	Open
	Files of type: All files (* *)	_	Cancel

After adding everything the folders should contain the following:



Go to the top toolbar then click on Execute > Compile & Run and if there are no problems the following output should appear:



Now that the code is up and running; open the ert_main.c and make the following modifications to the main function:

#if 0

```
printf("Warning: The simulation will run forever. "
```

"Generated ERT main won't simulate model step behavior. "

"To change this behavior select the 'MAT-file logging' option.\n");

fflush((NULL));

```
while (rtmGetErrorStatus(0) == (NULL)) {
```

/* Perform other application tasks here */

}

#else

int32_T Input_A = 1;

int32_T Input_B = 2;

while(1)

{

/* Pass input data */

Calc_Test_Model_U_Input_A = Input_A;

Calc_Test_Model_U_Input_B = Input_B;

/* Run Subsystem */

Calc_Test_Model_step(&Calc_Test_Model_DWork, Calc_Test_Model_U_Input_A, Calc_Test_Model_U_Input_B, &Calc_Test_Model_Y_Out1);

/* View Output */

printf("Input_A: %ld , Input_B: %ld\n\n"

"sum: %ld\n\n"
"sub: %ld\n\n"
"mul: %ld\n\n"
"div: %lf\n\n"
"sum_gain: %ld\n\n"
,
Calc_Test_Model_U_Input_A,
Calc_Test_Model_U_Input_B,
Calc_Test_Model_Y_Out1.sum,
Calc_Test_Model_Y_Out1.sub,
Calc_Test_Model_Y_Out1.mul,
Calc_Test_Model_Y_Out1.sum];

Input_A++;

Input_B++;

sleep(1000);

}

#endif

The modifications to the code add the Inputs A and B initialized to 1 and 2 respectively. Then an infinite while loop is started. Inside this loop the Inputs are passed to their corresponding structure elements located in the input variables for the model step function denoted by the _U (e.g. Calc_Test_Model_U_Input_A), next the step function, denoted _step (e.g. Calc_Test_Model_step) is called to run the model. After calling the step function the values are

now in the output data structure denoted by the $_Y$ (e.g.:Calc_Test_Model_Y_Out1) and then these can be accessed globally for any purpose.

The output data can then be viewed with the printf function as shown below:



9. Comparing output of MATLAB with that of the generated code

Once the code is generated and running there is a couple of ways to determine if the output is correct:

- Simple observation of the simulation output can be used in the case of simple calculations as those carried out in this example.
- In the case of more complex calculations a text file can be generated and plotted to compare to output data from the simulation model.
- Text files can be loaded into MATLAB (Ref. [1]) then the difference between these two files can be plotted showing differences in output for each sample.

References

Computer Software

[1] MATLAB, The MathWorks, Inc., Software Package, Ver. R2011b, Natick, Massachusetts, 2011.

[2] Simulink, The MathWorks, Inc., Software Package, Ver. R2011b, Natick, Massachusetts, 2011.

[3] Dev-C++, Bloodshed Software, Software Package, Ver. 5, 2005.

Appendix A Hardware Implementation Table

Key:	float and double (not listed) always equal 32 and 64, respectively										
	pointer (not listed) matches the value of int										
	Roun	Rounds to = Signed integer division rounds to									
	Shift	Shift right = Shift right on a signed integer as arithmetic shift									
Device vendor / Device type	Number of bits						st c size	Byte ordering	Round s to	Shift right	
	cha r	shor t	int	long	native	int	float				
Generic	-										
Unspecified (assume 32- bit Generic) (default)	8	16	32	32	32	Х	x	Unspecifie d	x	Set	
Custom	x	X	X	X	X	X	X	X	X	x	
16-bit Embedded Processor	8	16	16	32	16	X	X	X	X	Set	
32-bit Embedded Processor	8	16	32	32	32	x	X	X	X	Set	
32-bit Real Time Simulator	8	16	32	32	32	x	X	x	X	Set	
32-bit x86	8	16	32	32	32	X	X	Little	Zero	Set	

Key:	float and double (not listed) always equal 32 and 64, respectively												
	pointer (not listed) matches the value of int Rounds to = Signed integer division rounds to												
	Shift	Shift right = Shift right on a signed integer as arithmetic shift											
Device vendor / Device type	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right			
20000 Gpc	cha r	shor t	int	long	native	int	float						
compatible					-			Endian					
8-bit Embedded Processor	8	16	16	32	8	X	X	X	x	Set			
MATLAB Host Computer	8	16	32	Host specifi c value (32 or 64)	Host specifi c value (32 or 64)	x	x	Little Endian	x	Set			
AMD					·			·					
K5/K6/Athlo n	8	16	32	32	32	X	X	Little Endian	X	Set			
ARM Compatible													
ARM 7/8/9/10	8	16	32	32	32	Lon g	Float	X	X	x			
ARM 11	8	16	32	32	32	Lon g	Doubl e	x	X	X			

Key:	float and double (not listed) always equal 32 and 64, respectively												
	pointer (not listed) matches the value of int												
	Rour	Rounds to = Signed integer division rounds to											
	Shift right = Shift right on a signed integer as arithmetic shift												
Device vendor / Device type	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right			
51	cha r	shor t	int	long	native	int	float						
ARM Cortex	8	16	32	32	32	Char	None	x	X	x			
ASIC/FPGA													
ASIC/FPGA	NA	NA	N A	NA	NA	NA	NA	NA	NA	NA			
Analog Device	s	1	1	1	1	1	1		1				
Blackfin	8	16	32	32	32	Lon g	Doubl e	Little Endian	Zero	Set			
SHARC	32	32	32	32	32	Lon g	Doubl e	Big Endian	Zero	Set			
TigerSHARC	32	32	32	32	32	Lon g	Doubl e	Little Endian	Zero	Set			
Atmel													
AVR	8	16	16	32	8	x	X	Little Endian	Zero	Set			
Freescale	Freescale												

Key:	float and double (not listed) always equal 32 and 64, respectively													
	pointer (not listed) matches the value of int													
	Rounds to = Signed integer division rounds to													
	Shift right = Shift right on a signed integer as arithmetic shift													
Device vendor / Device type	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right				
	cha r	shor t	int	long	native	int	float							
32-bit PowerPC	8	16	32	32	32	Lon g	Doubl e	Big Endian	Zero	Set				
68332	8	16	32	32	32	X	X	Big Endian	X	Set				
68HC08	8	16	16	32	8	X	X	Big Endian	X	Set				
68HC11	8	16	16	32	8	X	X	Big Endian	X	Set				
ColdFire	8	16	32	32	32	X	X	Big Endian	Zero	Set				
DSP563xx (16-bit mode)	8	16	16	32	16	X	X	x	X	Set				
HC(S)12	8	16	16	32	16	X	X	Big Endian	X	Set				
MPC52xx, MPC5500, MPC55xx, MPC5xx, MPC7400, MPC7xxx, MPC82xx, MPC83xx,	8	16	32	32	32	Lon g	Doubl e	X	Zero	Set				

Key:	float and double (not listed) always equal 32 and 64, respectively													
	pointer (not listed) matches the value of int													
	Rounds to = Signed integer division rounds to													
	Shift right = Shift right on a signed integer as arithmetic shift													
Device vendor /	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right				
	cha r	shor t	int	long	native	int	float							
MPC86xx, MPC8xx														
MPC85xx	8	16	32	32	32	Lon g	Float	X	Zero	Set				
S12x	8	16	16	32	16	x	Х	Big Endian	X	Set				
Infineon														
C16x, XC16x	8	16	16	32	16	x	X	Little Endian	Zero	Set				
TriCore	8	16	32	32	32	X	X	Little Endian	X	Set				
Intel														
8051 Compatible	8	16	16	32	8	X	X	X	X	Clea r				
x86/Pentium	8	16	32	32	32	x	X	Little Endian	X	Set				

Key:	float and double (not listed) always equal 32 and 64, respectively													
	pointer (not listed) matches the value of int													
	Rounds to = Signed integer division rounds to													
	Shift right = Shift right on a signed integer as arithmetic shift													
Device vendor / Device type	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right				
Device type	cha r	shor t	int	long	native	int	float							
Microchip														
PIC18	8	16	16	32	8	X	X	Little Endian	Zero	Set				
dsPIC	8	16	16	32	16	X	X	Little Endian	Zero	Set				
NEC	1	1	1	1	1	1	1	1	1					
V850	8	16	32	32	32	X	X	X	X	x				
Renesas				·				•						
M16C	8	16	16	32	16	X	X	Little Endian	X	X				
M32C	8	16	x	32	X	x	X	Little Endian	X	x				
R8C/Tiny	8	16	16	32	16	x	X	Little Endian	X	X				
SH-2/3/4	8	16	32	32	32	X	X	X	X	x				

Key:	float and double (not listed) always equal 32 and 64, respectively													
	point	pointer (not listed) matches the value of int												
	Rounds to = Signed integer division rounds to													
	Shift right = Shift right on a signed integer as arithmetic shift													
Device vendor / Device type	Num	ber of	bits			Largest atomic size		Byte ordering	Round s to	Shift right				
	cha r	shor t	int	long	native	int	float							
SGI	SGI													
UltraSPARC IIi	8	16	32	32	32	X	X	Big Endian	X	Set				
STMicroelectro	onics	-	<u> </u>	<u>.</u>	<u>.</u>			·						
ST10/Super1 0	8	16	16	32	16	X	X	Little Endian	Zero	Set				
Texas Instrume	ents	1		1	1	1	1	1	1					
C2000	16	16	16	32	16	Int	None	X	Zero	Set				
C5000	16	16	16	32	16	Int	None	Big Endian	Zero	Set				
C6000	8	16	32	40	32	Int	None	X	Zero	Set				
MSP430	8	16	16	32	16	X	X	Little Endian	Zero	Set				
TMS470	8	16	32	32	X	X	x	X	x	X				